diplom Documentation

Реліз 0.1.0

Andriy Vasyltsiv

December 23, 2013

Зміст

Магістерська дипломна робота на тему:

Зміст:

АНОТАЦІЯ

1.1 АНОТАЦІЯ

Кваліфікаційна робота бакалавра: 72 с., 12 рис., 2 таблиці, 2 додатки, 16 джерел.

Дана бакалаврська робота присвячена розробці модуля пошуку дефектів реакції програм. Даний модуль призначений для аналізу програмного коду на предмет вразливостей та оцінки рівня їх небезпечності. Цей модуль може використовуватись як у рамках кафедри №21 ВІТІ НТУУ "КПІ", так і за її межами у інформаційно-телекомунікаційних вузлах ЗСУ.

Модуль пошуку дефектів реакції програм реалізований із застосуванням об'єктно-орієнтованої мови програмування C ++, PHP, CMS Joomla, бібліотеки libExploit, генератора парсерів BISON, лексичного аналізатора FLEX, системи керування базами даних MySQL

1.2 ANNOTATION

Qualifying work degree: 72 p., 12 img., 2 table, 2 add, 16 sources.

This undergraduate paper is devoted to developing search module defect response programs. This module is designed to analyze the code in terms of vulnerabilities and assessment of their danger. This module can be used as part of the department number 21 VITI KPI and outside military headquarters in the UAF.

Search module defect response programs implemented using object-oriented programming language C ++, PHP, CMS Joomla, librarie libExploit, parser generator BISON, FLEX, database management system MySQL.

Розділ 2

ВСТУП

Підготовка фахівців з нав'язування кібернетичного впливу та захисту вимагає напрацювання теоретичних знань і практичних навичок з використання інструментальних засобів дослідження потенційно-небезпечних дефектів реакції програм. На сьогодні існує низка засобів для здійснення аналізу та пошуку дефектів, та зазвичай вони є недостатньо інформативними та вимагають попередньої підготовки у використанні ними, тому існує необхідність у створенні більш інтерактивних, масштабованих та інформативних, що сприятиме процесу навчання. Актуальність роботи полягає у створенні програмно-технічних засобів пошуку та дослідження потенційно-небезпечних дефектів реакції програм.

Об'єкт дослідження – підготовка фахівців з аналізу та використання потенційно-небезпечних дефектів реакції програм.

Предмет дослідження – інструментальні засоби аналізу та використання потенційно-небезпечних дефектів реакції програм.

Мета роботи: Розробити програмно-технічний комплекс для відпрацювання практичних навичок фахівців з аналізу та використання потенційно-небезпечних дефектів реакції програм щодо переповнення.

Завдання:

1. Проаналізувати напрямки підготовки фахівців з нав'язування кібернетичного впливу;
2. Побудувати програмно-технічний комплекс дослідження цільових програм на переповнення буфера;
3. Проаналізувати існуючі метрики програмного коду на предмет можливості використання їх для оцінки потенційно вразливих ділянок вихідних текстів програм
4. Провести оцінку цінності підготовки фахівців з нав'язування кібернетичного впливу на основі переповнення буферу.

КЛАССИФІКАЦІЯ КІБЕРНЕТИЧНИХ АТАК НА ОСНОВІ ВІДДАЛЕНОГО ВИКОНАННЯ КОДУ

3.1 Кібернетична атака. Види. Кібернетичні атаки на основі віддаленого виконання коду

Виходячи зі змісту та ролі інформації у сучасному світі, американський дослідник М. Маклюен виводить цікаву тезу, що звучить так: "Істинно тотальна війна - це війна за допомогою інформації". Аналіз сучасних поглядів дозволяє вважати, що інформаційна боротьба (ІБ) у цілому є комплексом взаємопов'язаних і узгоджених за цілями, місцем і часом заходів, орієнтованих на досягнення інформаційної переваги. Вона є результатом нових інформаційних технологій. У наслідок їх застосування набули змін не тільки засоби збройної боротьби, але й стратегія, і тактика ведення сучасних воєн, з'явилися нові концепції ведення бойових дій у "інформаційному столітті", що враховують нові фактори вразливості сторін. Кібернетичний вплив – це сукупність дій, спрямованих на зміну порядку функціонування КІС. КА – це масовий кібернетичний вплив однієї сторони на іншу (рис.1.1.). На сьогоднішній час інформаційні технології впроваджуються у сфери діяльності людини. Не виключенням є військові організації, від рівня надійності яких напряму залежить національна безпека країни. Однак з стрімким розвитком виникає проблема захисту від комп'ютерних атак, техніка яких також постійно розвивається. Зростання можливостей щодо несанкціонованого одержання інформації, розширення за рахунок розвитку інформаційних технологій, а також зацікавлення у несанкціонованому одержанні інформації, поява додаткових каналів витоку інформації, передусім у процесі обробки інформації засобами електронно-обчислювальної техніки, використання нових методів і засобів несанкціонованого здобуття інформації значно ускладнили умови інформаційної безпеки, особливо в частині протидії технічній розвідці та попередження несанкціонованої модифікації інформації шляхом зараження її вірусами і програмними закладками різних видів і типів.

Забезпечення інформаційної безпеки в умовах, що постійно змінюються, вимагає постійного проведення: - фундаментальних та прикладних досліджень явищ і процесів у даній предметній області; - підготовки необхідної кількості підготованих і компетентних фахівців. Рекомендації Постанови Кабінету Міністрів України, нещодавніх конференцій та семінарів, присвячених проблемі підготовки кадрів з інформаційної безпеки зводяться до необхідності здійснення заходів: - збільшення чисельності фахівців, оскільки їх кількість не задовольняє існуючим потребам; -вдосконалення навчального процесу з метою підготовки висококваліфікованих фахівців, оскільки теорія і практика інформаційної безпеки безперервно та інтенсивно розвиваються і нові досягнення повинні якнайшвидше знайти відображення у навчальних планах і програмах; Наслідком цього процесу і стала поява певної модифікації та тенденції у системі підготовки та підвищення кваліфікації з інформаційної безпеки. Отже, виникають нові пріоритети підготовки фахівців в даній області: 1) Підготовка та перепідготовка фахівців, здатних ефективно вирішувати сучасні задачі ІБ в Україні; 2) Збільшення чисельності фахівців, які проходять підготовку та перепідготовку за напрямком інформаційної безпеки; 3) Об'єднання зусиль провідних освітніх, наукових колективів та адміністративних органів для вирішення практичних проблем ІБ; 4)

Створення та постійний розвиток наукових шкіл в області ІБ; 5) Створення умов для забезпечення режиму ІБ держави в цілому, регіонів, підприємств та окремих громадян.

Для підготовки фахівців ІБ можна виділити наступні підходи:

- при збереженні однакових сфер та об'єктів професійної діяльності повинні бути зазначені відмінності за видами професійної діяльності, оскільки вони впливають на характер знань та вмінь фахівця;
- повинен бути визначений склад базових дисциплін, які, як правило, необхідні для кожної спеціальності;
- необхідно посилити і диференціювати загальнопрофесійну підготовку фахівців за кожною спеціалізацією;
- склад і зміст спеціальних дисциплін за кожною спеціалізацією повинні охоплювати інформацію, що складає всі види таємниці, розкривати всі види, методи, засоби та технологію ЗІ, однак, при цьому необхідно враховувати специфіку професійної діяльності випускника.

Одним із підходів до підготовки військових фахівців, які здатні підтримувати безпеку у кібернетичному просторі – комунікаційному просторі, який охоплює комп'ютерні мережі та електронні пристрої, що використовуються для збереження, обробки та обміну інформацією є підготовка фахівців, що починається з формуваннння бази до кваліфікаційного рівня бакалавр, а саме проходження циклів підготовки. Підготовка фахівців з інформаційної безпеки (рис. 1.2.) починається у циклі професійної і практичної підготовки, де на вивчаємих дисциплінах майбутніми фахівцями вивчаються матеріали, для формування загальних знань з комп'ютерних наук, вивчаються мови програмування, технології створення програмних продуктів, архітектура комп'ютерних та операційних систем. Але щоб стати фахівцем, потрібна додаткова підготовка яка отримується після захисту кваліфікаційної роботи, майбутні фахівці розподіляються за спеціалізаціями підготовки, фахівці формуються відповідно до спеціалізації.

На етапі підготовки згідно спеціалізації вивчаються такі дисципліни як: • Експлуатація та бойове застосування програмних засобів інформаційної боротьби в комп'ютеризованих системах та мережах спеціального призначення, • Методологічні основи інформаційної боротьби в комп'ютеризованих системах та мережах спеціального призначення, • Технології побудови програмних засобів інформаційної боротьби та ін. На етапі підготовки згідно спеціалізації потрібно приділяти багато уваги практичній підготовці та інтерактивному навчанню. Для цього пропонується підхід до створення тренувального середовища із завідомо впровадженими вразливостями для нав'язування тестових впливів (рис. 1.3.). Це середовище може слугувати матеріалом для вивчення вищеназваних дисциплін, як для відпрацювання технології побудови програмних засобів інформаційної боротьби, для відпрацювання методології впливу на програмні продукти, а також використання програм для сканування вразливостей.

Необхідно виділити найбільш небезпечні вразливості для створення такої системи, кожна з уразливостей потребує детального вивчення, це окрема спеціалізація, яка рамках підготовки фахівця з інформаційної безпеки потребує глибокого дослідження. Досить розповсюдженим видом комп'ютерних атак на інформаційні системи є атака на переповнення буфера. Переповнення буфера було та залишається дуже важливою проблемою в аспекті безпеки програмного забезпечення. Переповнення буфера (англ. buffer overflow або англ. buffer overrun), це явище, при якому програма, під час запису даних в буфер, перезаписує дані за межами буфера (Рис.1.4). Це може викликати несподівану поведінку, включно з помилками доступу до даних, невірними результатами, збоєм програми або дірою в системі безпеці. Переповнення буфера може бути викликане недостатньою перевіркою вхідних даних. Воно є базою для багатьох уразливостей в програмних продуктах і може бути злонамірено використане.

Проблема переповнення буфера з роками тільки ускладнювалася, з'являлися типи атак, в результаті були розроблені принципово нові атаки на переповнення буфера. Оскільки значна частка програмного забезпечення створюється на мові C/C++, в якій немає вбудованих засобів обробки рядків - а саме контролю розподілу пам'яті, тому вони використовують небезпечний програмний код, що не перевіряє довжину буфера, у який записуються зовнішні дані, отримані ззовні, внаслідок чого можливість перезапису інших даних програми, включаючи код, що дозволяє змінити виконання програми, незалежно

коду. Атаки, в основному, здійснюються на програмні застосування, виконуються в привілейованому режимі, що дозволяє підняти рівень привілеїв для виконання шкідливого коду. Зробивши аналіз найбільш поширених прийомів техніки переповнення буферу можна зробити висновок що фахівцям в даній предметній області необхідно мати глибокі знання в таких дисциплінах як: - операційні системи(знання архітектури ОС Linux/Unix-like/WinX); - системне, мережеве програмування (низькорівневе); - теорії побудови компіляторів / інтерпретаторів – теорія кінцевих автоматів; - практичні навички використання засобів відлагодження та дослідження програмних продуктів;

1.2. Типові ситуації використання переповнення буфера.

Техніки використання уразливості через переповнення буфера залежать від архітектури, операційної системи і ділянки пам'яті (рис.1.5.). Існують наступні види переповнення буфера: - переповнення у стекові (stack smashing - зрив стеку), який полягає у перезаписі адреси повернення з вразливої функції, що призводить до виконання коду(існуючого або підготовленого зловмисником) за адресом, вказаним атакуючим; - переповнення в сегментах даних та динамічних областях (DATA, BSS, HEAP overflow), яке являє собою корекцію набору даних, керуючих алгоритмом програми, а також вказівників на функції, класи та управління. Технічно освідомлений і злонамірений користувач може використати стекове переповнення буфера так: 1. Для перезапису локальної змінної, змінивши тим самим перебіг програми на більш вигідний для нападника. 2. Для перезапису адреси повернення в стековому кадрі. Коли буде виконане повернення з функції, виконання програми відновиться за адресою, вказаною нападником (зазвичай це адреса буфера поля вводу). Такий спосіб найбільш розповсюджений в архітектурах, де стек росте донизу (наприклад в архітектурі x86). 3. Для перезаписів вказівника на функцію, або обробника винятків, що будуть виконані згодом. Із методом «трамполайнінга» (англ. trampolining, буквально «стрибки на батуті»), якщо адреса даних, яку ввів користувач, невідома, але їх місцезнаходження зберігається в регістрі, тоді адреса повернення може бути перезаписана на адресу опкоду, який виконає перехід до даних, введених користувачем. Якщо їх місцезнаходження зберігається в реєстрі R, тоді потрібен стрибок до опкоду для стрибка R, це викличе виконання користувацьких даних. Розташування необхідних опкодів або байтів в пам'яті може бути знайдено в динамічних бібліотеках або самій програмі. Адреси опкодів можуть змінюватися між застосуваннями і версіями ОС. Також варто зазначити, що наявність таких вразливостей зазвичай перевіряється за допомогою фазинга. Переповнення буфера в кучі використовується відмінно від переповнення в стеку. Пам'ять в кучі динамічно розподіляється застосуванням під час виконання і, зазвичай, містить дані програми. Використання відбувається через пошкодження даних особливим чином, що призводить до перезапису внутрішніх структур даних, таких як зв'язний список вказівників. Використання безпечних бібліотек. Проблема переповнення буфера характерна для мов програмування C та C++, бо вони не приховують деталі низькорівневого представлення буферів як контейнерів для типів даних. Таким чином, задля уникнення переповнення буфера, потрібно забезпечити високий рівень контролю зі створенням і зміною програмного коду, який здійснює керування буферами. Використання бібліотек абстрактних типів даних, які виконують централізоване автоматичне керування буферами і включають в себе перевірку на переповнення — один з підходів для запобігання переповнення буферу.

Рис.1.5. Схема використання вразливості переповнення буферу.

Два основних типи даних, які дозволяють здійснити переповнення буфера в цих мовах — рядки і масиви. Таким чином, використання бібліотек для рядків і спискових структур даних, які були розроблені для запобігання і/або виявлення переповнень буфера, дозволить уникнути уразливостей. Шелкод (Корисне навантаження) (англ. shellcode, код запуску оболонки) - це двійковий виконуваний код, який зазвичай передає управління командному процесору, наприклад '/ bin / sh' Unix Shell , command.com в MS-DOS і cmd.exe в операційних системах Microsoft Windows. Шелкод може бути використаний як корисне навантаження експлойту , що забезпечує зловмисникові доступ до командного інтерпретатра (англ.shell) в комп'ютерній системі.

> При експлуатації віддаленої уразливості шелкод може відкривати заздалегідь заданий порт TCP уразливого комп'ютера, через який буде здійснюватися подальший доступ до командної оболонці, такий код називається прив'язує до порту (англ. port binding shellcode). Якщо шелкод здійснює підключення до порту комп'ютера атакуючого, що проводиться з метою

обходу брандмауера або NAT , то такий код називається зворотної оболонкою (англ. reverse shell shellcode). Шелкод зазвичай впроваджується в пам'ять експлуатованої програми, після чого на нього передається управління шляхом переповнення стека , або при переповненні буфера в купі , або використовуючи атаки форматної рядка. Передача управління шелкоду здійснюється перезаписом адреси повернення в стеку адресою впровадженого шелкоду, перезаписом адрес викликаються функцій або зміною обробників переривань. Результатом цього є виконання шелкоду, який відкриває командний рядок для використання зломщиком.

Зломщики пишуть шелкоди часто використовуючи прийоми, що приховують їх атаку. Вони часто нам

Шелкод майже завжди містить рядок з ім'ям оболонки . Усі вхідні пакети містять такий рядок завжди розглядаються як підозрілі в очах СОВ. Також, деякі програми не приймають неалфавітних-цифровий введення (вони не приймають будь-що, крім a-z, AZ, 0-9, і кілька інших символів.) Для проходження через всі ці заходи спрямовані проти вторгнення, зломщики використовують шифрування , самомодіфікуючийся код , поліморфний код і алфавітно-цифровий код.

Розрізняють два типи шелкоду : А. локальний (впровадження та виконання можливе тільки за безпосереднього доступу зловмисника до КС) В. віддалений (атака з віддаленої машини). Локальний шелкод використовується зловмисником, коли він обмежений в правах комп'ютерної системи, але може використати вразливість в програмному забезпеченні даної системи з підняттям привілеїв до того рівня що і цільовий процес. Віддалений шелкод використовується хакерами для захоплення контролю над процесом на віддаленій машині в локальній мережі чи в інтернеті. У разі успішного його виконання, хакер матиме можливість отримати доступ над машиною по мережі. Віддалені шелкоди переважно використовують TCP/IP-зєднання для забезпечення атакуючого доступом до командної оболонки. Такий шелкод може бути класифікований по тому, як він встановлює з'єднання.: якщо код може встановити з'єднання, його називають "reverse shell" або "connect-back shellcode"(рис.1.6.), так як він з'єднується з машиною атакуючого. З іншого боку, якщо зловмисник потребує встановити з'єднання, то такий шелкод називають " bindshell", так як він відкриває і прослуховує порт, по якому зловмисник може під'єднатися і отримати контроль над системою. Є третій тип, менш використовуваний, має назву "socket-reuse shellcode". Цей тип шелкоду іноді використовується, коли експлойт встановлює з'єднання з вразливим процесом, який до його запуску не закрив з'єднання. Тому він може використати з'єднання для комунікації з хакером для віддаленого управління системою. Цей тип шелкоду є складним в реалізації так як він повинен виконати пошук вразливого процесу, який має відкрите з'єднання.

3.2 Підходи пошуку потенційно-вразливих до кібернетичного впливу програмних засобів

На сьогоднішній час існує низка засобів для аналізу програм як у вигляді вихідних текстів так і у двійковому коді. Для виявлення вразливостей захисту в програмах застосовують такі інструментальні засоби: - динамічні відлагоджувачі. Інструменти, які дозволяють виробляти налагодження програми в процесі її виконання. - статичні аналізатори (статичні відлагоджувачі). Інструменти, які використовують інформацію, накопичену в ході статичного аналізу досліджуваної програми. Статичні аналізатори вказують на ті місця в програмі, в яких можливо знаходиться помилка. Ці підозрілі фрагменти коду можуть, як містити помилку, так і не нести ніякої небезпеки для виконання програми. Наявність вихідних кодів програми істотно спрощує пошук вразливостей. Розглянемо декілька інструментів для аналізу вихідних текстів досліджуваних програм: Інструмент BOON , який на основі глибокого семантичного аналізу автоматизує процес сканування вихідних текстів на Сі в пошуках уразливих місць, здатних призводити до переповнення буферу. Він виявляє можливі дефекти, припускаючи, що деякі значення є частиною неявного типу з конкретним розміром буфера. CQual - інструмент аналізу для виявлення помилок в Сі-програмах. Програма розширює мову Сі додатковими обумовленими користувачем специфікаторами типу. Програміст коментує свою програму з відповідними специфікаторами, і cqual перевіряє помилки. Неправильні анотації вказують на потенційні помилки. Cqual може використовуватися, щоб виявити потенційну уразливість форматною рядка. MOPS - інструмент для пошуку

вразливостей в захисті в програмах на Сі. Його призначення: динамічне коректування, що забезпечує відповідність програми на Сі статичної моделі. MOPS використовує модель аудиту програмного забезпечення, яка покликана допомогти з'ясувати, чи відповідає програма набору правил, визначеному для створення безпечних програм. ITS4 . Простий інструмент, який статично переглядає вихідний Сі / Сі++ - код для виявлення потенційних вразливостей захисту. Він зазначає виклики потенційно небезпечних функцій, таких, наприклад, як strcpy / memcpy, і виконує поверхневий семантичний аналіз, намагаючись оцінити, наскільки небезпечний такий код, а так само дає поради щодо його поліпшення. RATS. Утиліта RATS (Rough Auditing Tool for Security) обробляє код, написаний на Сі / Сі + +, а також може обробити ще і скрипти на Perl, PHP і Python. RATS переглядає вихідний текст, знаходячи потенційно небезпечні звернення до функцій. Мета цього інструменту - не остаточно знайти помилки, а забезпечити обгрунтовані висновки, спираючись на які фахівець зможе вручну виконувати перевірку коду. RATS використовує поєднання перевірок надійності захисту від семантичних перевірок в ITS4 до глибокого семантичного аналізу в пошуках дефектів, здатних привести до переповнення буфера, отриманих з MOPS. Flawfinder . Як і RATS, це статичний сканер вихідних текстів програм, написаних на C/C++. Виконує пошук функцій, які найчастіше використовуються некоректно, присвоює їм коефіцієнти ризику (спираючись на таку інформацію, як передані параметри) і складає список потенційно вразливих місць, впорядковуючи їх за ступенем ризику. Всі ці інструменти схожі і використовують тільки лексичний і найпростіший синтаксичний аналіз. Тому результати, видані цими програмами, можуть містити до 100% помилкових повідомлень. Bunch - засіб аналізу та візуалізації програм на Сі, яке будує граф залежностей, що допомагає аудитору розібратися в модульній структурі досліджуваної програми. Frama-C - відкритий, інтегрований набір інструментів для аналізу вихідного коду на мові Сі. Набір включає ACSL (ANSI / ISO C Specification Language) - спеціальна мова, що дозволяє детально описувати специфікації функцій Сі, наприклад, вказати діапазон допустимих вхідних значень функції і діапазон нормальних вихідних значень.

Цей інструментарій допомагає виробляти такі дії:

- здійснювати формальну перевірку коду;
- шукати потенційні помилки виконання;
- провести аудит або рецензування коду;
- проводити реверс-інжиніринг коду для поліпшення розуміння структури програмного коду;
- генерувати формальну документацію.

Виходячи з опису даних інструментальних засобів можна привести порівняльну таблицю (табл.1.1.) функціональних можливостей та характеристик цих застосувань:

Також при відсутності вихідного тексту для аналізу програмного коду можна використовувати динамічні відлагоджувачі, які також можуть допомогти виявити помилки в коді, які допущені компілятором. Найпоширенішими відлагоджувачами на даний момент є SoftIce, OllyDebug, IDA Pro, GDB: SoftIce – всім відомий відлагоджувальник для ОС сімейства Windows, який працює в режимі ядра, що дозволяє відлагоджувати драйвера та різного роду сервіси що працюють в привілейованому режимі процесора. Працює в обхід MS Debugging API, що дуже ускладнює реалізацію захисту від відлагодження. SoftICE був спочатку розроблений компанією NuMega, яка включала його в пакет програм для швидкої розробки високопродуктивних драйверів під назвою Driver Studio, який пізніше був придбаний Compuware. OllyDebug – це 32-бітний відлагоджувальник працюючий в непривілейованому режимі процесора. Він має достатньо зручний інтерфейс та корисні функції які значно полегшують процес від лагодження. В OllyDBG вбудований спеціальний аналізатор, який розпізнає і візуально позначає процедури, цикли, константи і рядки, звернення до функцій API, параметри цих функцій і т.п. IDA Pro – це одночасно інтерактивний дизасемблер і відлагоджувальник. Він дозволяє отримати асемблерний текс, який може бути застосований для аналізу роботи програми. Варто зазначити, що вбудований відлагоджувальник доволі примітивний, працює через MS Debugging API (в NT) і через бібліотеку ptrace (в UNIX), що робить його легкорозпізнаємим для захисних механізмів. Сильною стороною цього продукту є саме дизасемблер, який на сьогодні генерує якісний вихідний текс. Окрім цього існує велика кількість плагінів під даний програмний продукт, що значно розширює його можливості. Також можливе на-

писання плагінів на скрипкових мовах – є підтримка Ruby, Python. GDB - GNU Debugger – основний відлагоджувальник під UNIX, орієнтований на зовсім інший тип мислення, аніж всі вищеперераховані відлагоджувальники. Це не просто інтерактивний відлагоджувальник – це модуль управління виконанням програм з гнучким і потужним інтерфейсом. Негативною стороною даного відлагоджувальника є відсутність аналізу захисних механізмів програм, тому у разі їх присутності в коді – процес відлагоджування стає неможливим. Наведемо порівняльну характеристику інструментальних засобів від лагодження програм (табл.1.2.).

Зробивши аналіз сучасних засобів дослідження програм можна зробити висновок, що ні один із існуючих засобів не виконує візуального представлення вразливостей та помилок програмного забезпечення, що в свою чергу вимагає від користувача високого рівня кваліфікації та достатнього досвіду роботи в даній предметній області. Тому для підготовки фахівців з кібернетичного впливу необхідно звернути увагу на візуалізацію помилок та вразливостей а також їхній вплив на подальше виконання програми. А враховуючи всю складність технік і об'єми дисциплін необхідних для реалізації цих технік можна зробити висновок що значну частину навантаження при освоєнні даного виду атак потрібно приділити на практичні заняття – а саме якісне засвоєння основ реалізації шляхом візуального дослідження вразливостей та впливу шкідливого коду на адресний простір процесу та хід його виконання. Це необхідне насамперед для того щоби майбутні фахівці могли з легкістю орієнтуватися в будь-яких інструментах дослідження програмних застосувань та реалізації атак.

НАУКОВО-МЕТОДОЛОГІЧНІ ЗАСАДИ ВИЯВЛЕННЯ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ДЕФЕКТІВ РЕАКЦІЇ ПРОГРАМ НА ОСНОВІ АНАЛІЗУ ВИХІДНИХ ТЕКСТІВ

4.1 Методи та способи аналізу вихідних текстів програм на предмет наявності потенційно небезпечних дефектів реакції програм. Метрики програмного забезпечення.

На відміну від більшості галузей матеріального виробництва, в питаннях проектів створення ПЗ неприпустимі прості підходи, засновані на множенні трудомісткості на середню продуктивність праці. Це викликано, насамперед, тим, що економічні показники проекту нелінійно залежать від обсягу робіт, а при обчисленні трудомісткості допускається велика похибка.

Тому для вирішення цього завдання використовуються комплексні і досить складні методики, які вимагають високої відповідальності в застосуванні і певного часу на адаптацію (настройку коефіцієнтів).

Сучасні комплексні системи оцінки характеристик проектів створення ПЗ можуть бути використані для вирішення наступних завдань:

- попередня, постійна і підсумкова оцінка економічних параметрів проекту: трудомісткість, тривалість, вартість;
- оцінка ризиків по проекту: ризик порушення строків та невиконання проекту, ризик збільшення трудомісткості на етапах налагодження та супроводження проекту та пр;
- прийняття оперативних управлінських рішень - на основі відстеження певних метрик проекту можна своєчасно попередити виникнення небажаних ситуацій і усунути наслідки непродуманих проектних рішень.

4.1.1 Метрики

Якість ПЗ - це сукупність властивостей , які визначають корисність вироби (програми) для користувачів відповідно з функціональним призначенням і пред'явленими вимогами . Характеристика якості програми - поняття, що відображає окремі фактори , що впливають на якість програм і піддаються виміру .

Критерій якості - чисельний показник , що характеризує ступінь , в якій програмі притаманне оцінювана властивість .

Критерії якості включають такі характеристики: економічність , документованість , гнучкість , модульність , надійність , обгрунтованість , тестовані , ясність , точність , модифицируемость , ефективність , легкість супроводу і т.д.

Критерій повинен :

- Чисельно характеризувати основну цільову функцію програми ;
- Забезпечувати можливість визначення витрат , необхідних для досягнення необхідного рівня якості , а також ступеня впливу на показник якості різних зовнішніх факторів;
- Бути по можливості простим , добре вимірним і мати малу дисперсію .

Для вимірювання характеристик і критеріїв якості використовують метрики .

Метрика якості програм - система вимірювань якості програм . Ці виміри можуть проводитися на рівні критеріїв якості програм або на рівні окремих характеристик якості. У першому випадку система вимірювань дозволяє безпосередньо порівнювати програми за якістю. При цьому самі виміри не можуть бути проведені без суб'єктивних оцінок властивостей програм. У другому випадку вимірювання характеристик можна виконати об'єктивно і достовірно , але оцінка якості ПЗ в цілому буде пов'язана з суб'єктивною інтерпретацією одержуваних оцінок.

У дослідженні метрик ПО розрізняють два основних напрямки:

- Пошук метрик , що характеризують найбільш специфічні властивості програм, тобто метрик оцінки самого ПЗ;
- Використання метрик для оцінки технічних характеристик і факторів розробки програм, тобто метрик оцінки умов розробки програм.

По виду інформації , одержуваної при оцінці якості ПО метрики можна розбити на три групи:

- Метрики , що оцінюють відхилення від норми характеристик вихідних проектних матеріалів . Вони встановлюють повноту заданих технічних характеристик вихідного коду.
- Метрики , що дозволяють прогнозувати якість розроблюваного ПЗ. Вони задані на безлічі можливих варіантів рішень поставленого завдання і їх реалізації і визначають якість ПЗ , яке буде досягнуто в результаті.
- Метрики , за якими приймається рішення про відповідність кінцевого ПО заданим вимогам . Вони дозволяють оцінити відповідність розробки заданим вимогам .

4.1.2 ОСНОВНІ НАПРЯМКИ ЗАСТОСУВАННЯ МЕТРИК

В даний час у світовій практиці використовується кілька сотень метрик програм. Існуючі якісні оцінки програм можна згрупувати по шести напрямках :

- Оцінки топологічної та інформаційної складності програм;
- Оцінки надійності програмних систем , що дозволяють прогнозувати отказовие ситуації ;
- Оцінки продуктивності ПО і підвищення його ефективності шляхом виявлення помилок проектування ;
- Оцінки рівня мовних засобів і їх застосування;
- Оцінки труднощі сприйняття і розуміння програмних текстів , орієнтовані на психологічні фактори , суттєві для супроводу і модифікації програм;
- Оцінки продуктивності праці програмістів для прогнозування термінів розробки програм і планування робіт зі створення програмних комплексів.

4.1.3 МЕТРИЧНІ ШКАЛИ

Залежно від характеристик і особливостей застосовуваних метрик їм ставляться у відповідність різні вимірювальні шкали.

Номінальною шкалою відповідають метрики , що класифікують програми на типи за ознакою наявності або відсутності деякої характеристики без урахування градацій.

Порядкової шкалою відповідають метрики , що дозволяють ранжувати деякий характеристики шляхом порівняння з опорними значеннями , тобто вимір за цією шкалою фактично визначає взаємне положення конкретних програм.

Інтервального шкалою відповідають метрики , які показують не тільки відносне положення програм , але і те , як далеко вони відстоять один від одного.

Відносній шкалі відповідають метрики , що дозволяють не тільки розташувати програми певним чином і оцінити їх положення відносно один одного , а й визначити , як далеко оцінки відстоять від кордону , починаючи з якої характеристика може бути виміряна.

Метрика СКЛАДНОСТІ ПРОГРАМ

При оцінці складності програм , як правило , виділяють три основні групи метрик :

- Метрики розміру програм
- Метрики складності потоку управління програм
- І метрики складності потоку даних програм

. Метрика РОЗМІРУ ПРОГРАМ.

Оцінки першої групи найбільш прості і , очевидно , тому отримали широке поширення. Традиційною характеристикою розміру програм є кількість рядків вихідного тексту. Під рядком розуміється будь-який оператор програми , оскільки саме оператор , а не окремо взята рядок є тим інтелектуальним " квантом " програми, спираючись на який можна будувати метрики складності її створення. Безпосереднє вимірювання розміру програми , незважаючи на свою простоту , дає хороші результати. Звичайно , оцінка розміру програми недостатня для прийняття рішення про її складності , але цілком застосовна для класифікації програм , істотно різняться обсягами. При зменшенні відмінностей в обсязі програм на перший план висуваються оцінки інших факторів, що впливають на складність . Таким чином , оцінка розміру програми є оцінка за номінальною шкалою , на основі якої визначаються тільки категорії програм без уточнення оцінки для кожної категорії .

До групи оцінок розміру програм можна віднести також і метрику Холстеда .

МЕТРИКА Холстеда . Основу метрики Холстеда складають чотири вимірюваних характеристики програми : n1 - число унікальних операторів програми , включаючи символи - роздільники , імена процедур і знаки операцій (словник операторів) ; n2 - число унікальних операндів програми (словник операндів) ; N1 - загальне число операторів в програмі ; N2 - загальне число операндів в програмі. Спираючись на ці характеристики , одержувані безпосередньо при аналізі вихідних текстів програм , М. Холстед вводить такі оцінки:

словник програми

n1 = n1 + n2 , довжину програми N = N1 + N2 , (1) обсяг програми V = N * log2 (n) (біт). (2) Під бітом мається на увазі логічна одиниця інформації - символ , оператор , операнд . Далі М. Холстед вводить n * - теоретичний словник програми, тобто словниковий запас, необхідний для написання програми , з урахуванням того , що необхідна функція вже реалізована в даній мові і, отже , програма зводиться до виклику цієї функції. Наприклад , згідно М. Холстеду , можливе здійснення процедури виділення простого числа могло б виглядати так :

CALL SIMPLE (X , Y) , де Y - масив чисельних значень , що містить шукане число X. Теоретичний словник в цьому випадку буде складатися з n1 * : { CALL , SIMPLE (...) } , n1 * = 2 ; n2

* : { X , Y } , n2 * = 2 , а його довжина , обумовлена як n * = n1 * + n2 * , буде дорівнювати 4 . Використовуючи n * , Холстед вводить оцінку V * : V * = n ** log2 n * , (3) за допомогою якої описується потенційний обсяг програми , відповідний максимально компактному тексту програми, що реалізує даний алгоритм .

МЕТРИКА СКЛАДНОСТІ ПОТОКУ УПРАВЛІННЯ ПРОГРАМ.

Друга найбільш представницька група оцінок складності програм - метрики складності потоку керування програм. Як правило , за допомогою цих оцінок оперують або щільністю керуючих переходів усередині програм, або взаємозв'язками цих переходів .

І в тому і в іншому випадку стало традиційним уявлення програм у вигляді керуючого орієнтованого графа G = (V , E) , де V - вершини , відповідні операторам , а E - дуги , відповідні переходам .

МЕТРИКА МакКейб . Вперше графічне представлення програм було запропоновано МакКейб . Основний метрикою складності він пропонує вважати цикломатическая складність графа програми , або , як її ще називають , цикломатичне число МакКейб , що характеризує трудомісткість тестування програми . Для обчислення цикломатическая числа МакКейб Z (G) застосовується формула

Z (G) = e - v +2 p , де e - число дуг орієнтованого графа G ; v - число вершин ; p - число компонентів зв'язності графа. Число компонентів зв'язності графа можна розглядати як кількість дуг , які необхідно додати для перетворення графа в сильно зв'язний . Сильний зв'язковим називається граф , будь-які дві вершини якого взаємно досяжні. Для графів коректних програм , тобто графів , що не мають недосяжних від точки входу ділянок і " висячих " точок входу і виходу , сильно зв'язний граф , як правило , виходить шляхом замикання дугою вершини , що позначає кінець програми , на вершину , що позначає точку входу в цю програму.

По суті Z (G) визначає число лінійно незалежних контурів у Сильно зв'язкового графі . Інакше кажучи , цикломатичне число МакКейб показує необхідну кількість проходів для покриття всіх контурів сильно зв'язного графа або кількість тестових прогонів програми , необхідних для вичерпного тестування за критерієм " працює кожна гілка" .

Для програми , граф якої зображений на рис.1 , цикломатичне число при e = 10 , v = 8 , p = 1 визначиться як Z (G) = 10-8 +2 = 4 .

Цикломатичне число залежить тільки від кількості предикатів , складність яких при цьому не враховується. Наприклад , є два оператора умови:

IF X > 0 THEN X = A ; ELSE ; і IF (X > 0 & FLAG = '1 'B) ! (X = 0 & FLAG = '0 'B) THEN X = A ; ELSE ; Обидва оператора припускають єдине розгалуження і можуть бути представлені одним і тим же графом (рис. 2) . Очевидно , цикломатичне число буде для обох операторів однаковим , що не відображає складності предикатів , що досить істотно при оцінці програм.

МЕТРИКА Майерс . Виходячи з цього Г. Майерс запропонував розширення цієї метрики . Суть підходу Г. Майерса полягає в представленні метрики складності програм у вигляді інтервалу [Z (G) , Z (G) + h] . Для простого предиката h = 0 , а для n -місних предикатів h = n - 1 . Таким чином , перший оператору відповідає інтервал [2,2] , а другий - [2,6] . По ідеї така метрика дозволяє розрізняти програми , представлені однаковими графами. На жаль , інформація про результати використання цього методу відсутня , тому нічого не можна сказати про його застосовності .

МЕТРИКА ПІДРАХУНКУ точок перетину. Розглянемо метрику складності програм , що отримала назву " підрахунок точок перетину " , авторами якої є М. Вудвард , М. Хенель і Д. Хидли . Метрика орієнтована на аналіз програм , при створенні яких використовувалося неструктурні

кодування на таких мовах , як мова асемблера і Фортран . У графі програми , де кожному оператору відповідає вершина , тобто не виключені лінійні ділянки , при передачі управління від вершини a до b номер оператора a дорівнює min (a , b) , а номер оператора b - max (a , b) . Точка перетину дуг з'являється , якщо

min (a , b) < min (p , q) < max (a , b) & max (p , q) > max (a , b) | min (a , b) < max (p , q) < max (a , b) & min (p , q) < min (a , b) . Іншими словами , точка перетину дуг виникає у разі виходу управління за межі пари вершин (a , b) (рис. 3) . Кількість точок перетину дуг графа програми дає характеристики не структурованості програми .

МЕТРИКА Джілбі . Однією з найбільш простих , але , як показує практика , досить ефективних оцінок складності програм є метрика Т. Джілбі , в якій логічна складність програми визначається як насиченість програми виразами типу IF- THEN - ELSE . При цьому вводяться дві характеристики : CL - абсолютна складність програми , що характеризується кількістю операторів умови; cl - відносна складність програми , що характеризується насиченістю програми операторами умови, тобто cl визначається як відношення CL до загального числа операторів. Використовуючи метрику Джілбі , ми доповнили її ще однією складовою , а саме характеристикою максимального рівня вкладеності оператора CLI , що дозволило не тільки уточнити аналіз по операторам типу IF- THEN - ELSE , але й успішно застосувати метрику Джілбі до аналізу циклічних конструкцій.

МЕТРИКА ГРАНИЧНИХ ЗНАЧЕНЬ Великий інтерес представляє оцінка складності програм за методом граничних значень . Введемо кілька додаткових понять , пов'язаних з графом програми .

Нехай G = (V , E) - орієнтований граф програми з єдиною початковою і єдиною кінцевою вершинами. У цьому графі число входять вершин у дуг називається негативною ступенем вершини , а число що виходять з вершини дуг - позитивної ступенем вершини . Тоді набір вершин графа можна розбити на дві групи: вершини , у яких позитивна ступінь <= 1 ; вершини , у яких позитивна ступінь > = 2 .

Вершини першої групи назвемо приймаючими вершинами , а вершини другої групи - вершинами відбору.

Для отримання оцінки за методом граничних значень необхідно розбити граф G на максимальне число подграфов G ' , що задовольняють таким умовам: вхід в подграф здійснюється тільки через вершину відбору; кожен подграф включає вершину (звану надалі нижньою межею подграфа) , в яку можна потрапити з будь іншої вершини подграфа . Наприклад , вершина відбору , поєднана сама з собою дугою - петлею , утворює подграф . (рис. 4 , таблиця 1) .

Число вершин , що утворюють такий подграф , одно скоригованої складності вершини відбору (таблиця 2) . Кожна приймаюча вершина має скориговану складність , рівну 1 , крім кінцевої вершини , скоригована складність якої дорівнює 0 . Скориговані складності всіх вершин графа G підсумовуються , утворюючи абсолютну граничну складність програми . Після цього визначається відносна гранична складність програми :

S0 = 1 - (v - 1) / Sa , де S0 - відносна гранична складність програми ; Sa - абсолютна гранична складність програми ; v - загальне число вершин графа програми .

Метрика СКЛАДНОСТІ ПОТОКУ ДАНИХ . Інша група метрик складності програм - метрики складності потоку даних , тобто використання , конфігурації і розміщення даних в програмах. МЕТРИКА ОБІГУ до глобальних змінних . Розглянемо метрику , що зв'язує складність програм із зверненнями до глобальних змінних . Пара " модуль - глобальна змінна" позначається як (p , r), де p - модуль , що має доступ до глобальної змінної r . Залежно від наявності в програмі реального обігу до змінної r формуються два типи пар " модуль - глобальна змінна" : фактичні і можливі . Можливе звернення до r за допомогою p показує , що область існування r включає в себе p .

Характеристика Aup говорить про те , скільки разів модулі Up дійсно отримували доступ до глобальних змінних , а число Pup - скільки разів вони могли б отримати доступ.

Відношення числа фактичних звернень до можливих визначається

Rup = Aup / Pup . Ця формула показує наближену ймовірність посилання довільного модуля на довільну глобальну змінну. Очевидно , чим вище ця вірогідність , тим вище ймовірність " несанкціонованого " зміни якої-небудь змінної , що може істотно ускладнити роботи, пов'язані з модифікацією програми . На жаль , поки не можна сказати , наскільки зручний і точний цей метод на практиці , так як немає відповідних статистичних даних.

МЕТРИКА Спен Визначення Спен грунтується на локалізації звернень до даних всередині кожної програмної секції. Спен - це число тверджень , які містять даний ідентифікатор , між його першим і останнім появою в тексті програми . Отже , ідентифікатор , що з'явився n раз , має Спен , рівний n - 1 . При великому Спен ускладнюється тестування і налагодження . 1

МЕТРИКА ЧЕПІНА . Суть методу полягає в оцінці інформаційної міцності окремо взятого програмного модуля за допомогою аналізу характеру використання змінних зі списку вводу-виводу. Всі безліч змінних , складових список введення-виведення , розбивається на 4 функціональні групи:

1 . P - що вводяться змінні для розрахунків та для забезпечення виведення .

Прикладом може служити використовувана в програмах лексичного аналізатора змінна, що містить рядок вихідного тексту програми, тобто сама змінна не модифікується , а лише містить вихідну інформацію .

2 . M - модифікуються , або створювані всередині програми змінні.

3 . C - змінні , що беруть участь в управлінні роботою програмного модуля (керуючі змінні).

4 . T - які не використовуються в програмі (" паразитні ") змінні. Оскільки кожна змінна може виконувати одночасно кілька функцій , необхідно враховувати її в кожній відповідній функціональній групі .

Далі вводиться значення метрики Чепіна :

Q = a1 * P + a2 * M + a3 * C + a4 * T , (4) де a1 , a2 , a3, a4 - вагові коефіцієнти .

Вагові коефіцієнти у виразі (4) використані для відбиття різного впливу на складність програми кожної функціональної групи . На думку автора метрики , найбільшу вагу , що дорівнює трьом , має функціональна група C , так як вона впливає на потік управління програми . Вагові коефіцієнти решти груп розподіляються наступним чином : a1 = 1 , a2 = 2 , a4 = 0.5 . Ваговий коефіцієнт групи T НЕ дорівнює 0 , оскільки " паразитні " змінні не збільшують складність потоку даних програми , але іноді ускладнюють її розуміння . З урахуванням вагових коефіцієнтів вираз (4) приймає вигляд:

Q = P + 2M + 3C + 0.5T Слід зазначити , що розглянуті метрики складності програм засновані на аналізі вихідних текстів програм і графів , що забезпечує єдиний підхід до автоматизації з розрахунку .

Метрика стилістики та зрозумілої програми

МЕТРИКА РІВНЯ коментування програм . Найбільш простий метрикою стилістики та зрозумілості програм є оцінка рівня коментування програми F : F = Nком / Nстр , (5) де Nком - кількість коментарів у програмі ; Nстр - кількість рядків або операторів вихідного тексту .

Таким чином , метрика F відображає насиченість програми коментарями.

Виходячи з практичного досвіду прийнято вважати , що F> = 0.1 , тобто на кожні десять рядків програми має припадати мінімум один коментар . Як показують дослідження , коментарі розподіляються по тексту програми нерівномірно: на початку програми їх надлишок , а в середині або в кінці - недолік. Це пояснюється тим , що на початку програми , як правило , розташовані оператори опису ідентифікаторів , що вимагають більш " щільного "

коментування . Крім того , на початку програми також розташовані " шапки" , що містять загальні відомості про виконавця , характері , функціональне призначення програми і т. п. Така насиченість компенсує недолік коментарів у тілі програми , і тому формула (5) недостатньо точно відображає коментування функціональної частини тексту програми .

Більш вдалий варіант , коли вся програма розбивається на n рівних сегментів і для кожного з них визначається Fi:

Fi = sign (Nком / Nстр - 0.1) , при цьому

n F = Сума (Fi) . i = 1 Рівень коментування програми вважається нормальним , якщо виконується умова: F = n . В іншому випадку небудь фрагмент програми доповнюється коментарями до номінального рівня .

Метрика Холстедом Наступні п'ять характеристик є продовженням метрики Холстеда . 1 . Для вимірювання теоретичної довжини програми N ^ M. Холстед вводить аппроксимирующую формулу:

N ^ = n1 * log2 (n1) + n2 * log2 (n2) , (6) де n1 - словник операторів; n2 - словник операндів програми .

Вводячи цю оцінку , Холстед виходить з основних концепцій теорії інформації , за аналогією з якими частота використання операторів і операндів у програмі пропорційна двійковому логарифму кількості їх типів . Таким чином , вираз (6) являє собою ідеалізовану апроксимацію (1), тобто справедливо для потенційно коректних програм , вільних від надмірності або недосконалостей (стилістичних помилок). Недосконалостями можна вважати такі ситуації:

а) наступна операція знищує результати попередньої без їх використання;

б) присутні тотожні вирази , вирішальні абсолютно однакові завдання ;

в) однієї і тієї ж змінної призначаються різні імена і т. п.

Подібні ситуації приводять до зміни N без зміни n .

М. Холстед стверджує , що для стилістично коректних програм відхилення в оцінці теоретичної довжини N ^ від реальної N не перевищує 10 %.

Ми пропонуємо використовувати N ^ як еталонне значення довжини програми зі словником n . Довжина коректно складеної програми N , тобто програми , вільної від надмірності і має словник n , не повинна відхилятися від теоретичної довжини програми N ^ більш ніж на 10 %. Таким чином , вимірюючи n1 , n2 , N1 і N2 і зіставляючи значення N і N ^ для деякої програми , при більш ніж 10 % - не відхилення можна говорити про наявність в програмі стилістичних помилок , тобто недосконалостей .

На практиці N і N ^ часто істотно розрізняються.

2 . Іншою характеристикою , що належить до метрикам коректності програм , по М. Холстеду , є рівень якості програмування L (рівень програми):

L = V * / V , (7) де V і V * визначається відповідно виразами (2) і (3) .

Вихідним для введення цієї характеристики є припущення про те , що при зниженні стилістичного якості програмування зменшується змістовне навантаження на кожен компонент програми і, як наслідок , розширюється обсяг реалізації вихідного алгоритму . Враховуючи це, можна оцінити якість програмування на підставі ступеня розширення тексту щодо потенційного обсягу V * . Очевидно , для ідеальної програми L = 1 , а для реальної - завжди L < 1 .

3 . Нерідко доцільно визначити рівень програми , не вдаючись до оцінки її теоретичного обсягу , оскільки список параметрів програми часто залежить від реалізації і може бути штучно розширений. Це призводить до збільшення метричної характеристики якості програмування. М. Холстед

пропонує апроксимувати цю оцінку виразом , що включає тільки фактичні параметри , тобто параметри реальної програми :

L ^ = 2 * n2 / (n1 * N2) . 4 . Маючи в своєму розпорядженні характеристикою L ^ , Холстед вводить характеристику I , яку розглядає як інтелектуальний зміст конкретного алгоритму , інваріантне по відношенню до використовуваних мов реалізації : I = L ^ * V. (8)

На наш погляд, та й на думку самого автора , термін інтелектуальність не зовсім вдалий. Перетворюючи вираз (8) з урахуванням (7) , отримуємо

I = L ^ V = LV = V * V / V = V * . Еквівалентність I і V * свідчить про те , що ми маємо справу з характеристикою інформативності програми .

Введення характеристики I дозволяє визначити розумові витрати на створення програми . Процес створення програми умовно можна представити як ряд операцій:

1) осмислення пропозиції відомого алгоритму ;

2) запис пропозиції алгоритму в термінах використовуваної мови програмування, тобто пошук в словнику мови відповідної інструкції , її змістове наповнення і запис .

Використовуючи цю формалізацію в методиці Холстеда , можна сказати , що написання програми по заздалегідь відомим алгоритмом є N ^ -разова вибірка операторів і операндів зі словника програми n , причому число порівнянь (за аналогією з алгоритмами сортування) складе log2 (n) .

Якщо врахувати , що кожна вибірка - порівняння містить , в свою чергу , ряд уявних елементарних рішень , то можна поставити у відповідність змістовної навантаженні кожної конструкції програми складність і число цих елементарних рішень . Кількісно це можна характеризувати за допомогою характеристики L , оскільки 1 / L має сенс розглядати як середній коефіцієнт складності , що впливає на швидкість вибірки для даної програми . Тоді оцінка необхідних інтелектуальних зусиль з написання програми може бути виміряна як

E = N ^ * log2 (n / L). (9) Таким чином , E характеризує число необхідних елементарних рішень при написанні програми .

Однак слід зауважити , що E адекватно характеризує лише початкові зусилля з написання програм , оскільки при побудові E не враховуються налагоджувальні роботи , які вимагають інтелектуальних витрат іншого характеру.

Суть інтерпретації цієї характеристики полягає в оцінці не витрат на розробку програми , а витрат на сприйняття готової програми . При цьому замість теоретичної довжини програми N ^ використовується її реальна довжина:

E ‘ = N * log2 (n / L). Характеристика E ‘ введена виходячи мз припущення , що інтелектуальні зусилля на написання і сприйняття програми дуже близькі за своєю природою. Однак якщо при написанні програми стилістичні похибки в тексті практично не повинні відображатися на інтелектуальній трудомісткості процесу , то при спробі зрозуміти таку програму їх присутність може привести до серйозних ускладнень. Ця посилка досить добре узгоджується з нашими висновками щодо взаємозв’язку N і N ^ , викладеними вище.

Перетворюючи формулу (9) з урахуванням виразів (2) і (7) , отримуємо

E = V * V / V * . Таке уявлення E ‘ , а відповідно і E , так як E = E’ , наочно ілюструє доцільність розбиття програм на окремі модулі , оскільки інтелектуальні витрати виявляються пропорційними квадрату обсягу програми , який завжди більше суми квадратів обсягів окремих модулів.

МЕТРИКА ЗМІНИ ДОВЖИНИ програмної документації . Розглянемо ще одну метрику , за своїм характером кілька відрізняється від попередніх. Вона спирається на принцип оцінки , при якому використовується вимірювання флуктуації довжин програмної документації .

Вихідним є припущення про те , що чим менше змін і коригувань вноситься в програмну документацію , тим більш чітко були сформульовані розв'язувані завдання на всіх етапах робіт . На думку автора метрики , неточності і неясності при створенні ПЗ служать причиною збільшення кількості коригувань та змін в документації. І, навпаки , демпфовані перехідний процес з нечисленними змінами довжин документів - природний наслідок добре обдуманої ідеї , добре проведеного аналізу, проектування і ясною структури програм. Ці взаємозв'язки і є основними для даного методу оцінки , суть якого полягає в наступному.

Припустимо , що документація змінюється в дискретні моменти часу t (i) , i = 1,2 , ... , n . Тоді в будь-який момент часу t (i) поточна довжина документа l (i) може бути визначена як

l (i) = l (i - 1) + a (i) - b (i) ; l (0) = 0 , де l (i - 1) - довжина документа в попередній момент часу; a (i) - добавляемая частина документа; b (i) - исключаемая частину документа. Далі вводиться d (i), що представляє собою відхилення поточної довжини документа l (i) від кінцевого значення l (n) :

d (i) = l (n) - l (i) . Потім розраховується інтеграл по модулю цього відхилення на інтервалі від t (i) до t (n), представлений у вигляді суми :

n - 1 H (n) = Сума | d (i) | * (t (i +1) - t (i)) . (10) i = 1 Значення H (n) являє собою оцінку перехідного процесу для інтервалу часу від t (1) до t (n) . Однак H (n) не враховує змін типу a (i) = b (i) , хоча вони , безперечно , впливають на хід подальшого процесу .

Щоб відобразити вплив змін такого роду , які називаються в подальшому імпульсними , вводиться експонентна функція , що відображає функцію відгуку. Заштрихованная область на рис.5 являє собою додаток до оцінки H , що відбиває вплив імпульсного зміни довжини документів і обчислюється як

Інтеграл a (i) * e ^ (-L ^ (-1 * (tt (i)))) dt = L * L (i) = L * b (i) , L> 0 . (11) t (i)

Таким чином , оцінка довжини документа пропорційна значенню імпульсного зміни довжини a (i) = b (i) з коефіцієнтом пропорційності L.

В принципі імпульсна зміна довжини документа присутній і при a (i) < > b (i) . Тому з урахуванням (11) автор метрики перетворює вираз (10) до виду

n - 1 H ' (n) = Сума [| d (i) | * (t (i +1) - t (i)) + L * c (i)] , (12) i = 1 причому c (i) = min { a (i) , b (i) } .

Якщо в процесі роботи значення a (i) і b (i) неконтрольовані, імпульсна зміна довжини врахувати не можна. Тоді c (i) = 0 , і вираз (12) вироджується в (10). Використовуючи кінцеве значення довжини документа , можна записати

H (n) '' = H (n) '/ l (n) .

4.1.4 Моделі та метрики оцінки якості ПЗ

Сучасна програмна індустрія за півстоліття шукань накопичила значну колекцію моделей і метрик , що оцінюють окремі виробничі та експлуатаційні властивості ПЗ. Однак гонитва за їх універсальністю , неврахування області застосування розроблюваного ПЗ , ігнорування етапів життєвого циклу програмного забезпечення і, нарешті , необгрунтоване їх використання в різнопланових процедурах прийняття виробничих рішень , істотно підірвало до них довіру розробників і користувачів ПЗ. Проте , аналіз технологічного досвіду лідерів виробництва ПО показує , наскільки дорого обходиться недосконалість ненаукового прогнозу разрешимости і трудовитрат , складності програм , негнучкість контролю та управління їх розробкою та багато іншого , що вказує на відсутність наскрізної методичної підтримки і призводить зрештою до його невідповідності вимогам користувача , необхідному стандарту і до подальшої болючою і трудомісткою його переробці. Ці обставини , вимагають ретельного відбору методик , моделей , методів оцінки якості ПЗ , врахування обмежень їх придатності для різних життєвих

циклах і в межах життєвого циклу , встановлення порядку їх спільного використання , застосування надмірної разномодельного дослідження одних і тих же показників для підвищення достовірності поточних оцінок , накопичення та інтеграції різнорідної метричної інформації для прийняття своєчасних виробничих рішень і заключної сертифікації продукції . Нижче , в таблиці 1.3. , Наведені моделі та метрики , що добре зарекомендували себе при оцінці якості ПЗ, придатні для прогнозування та констатації різних показників складності і надійності програм.

4.1.5 Метрики СКЛАДНОСТІ

метрики Холстеда

- довжина програми ;
- Обсяг програми
- Оцінка її реалізації;
- Труднощі її розуміння ;
- Трудомісткість кодування ;
- Рівень мови вираження;
- Інформаційний зміст ;
- Оптимальна модульність ;

N = n1 log1 n1 + n2 log2 n2 V = N log2 n L * = (2 n2) / (n1 N2) Ec = V / L * D = (n1N2) (2n2) = 1 / L * l * = V / D2 = V / L * 2 I = V / D M = n2 * / 6

метрики Джілбі

- Кількість операторів циклу;
- Кількість операторів умови;
- Число модулів або підсистем ;
- Відношення числа зв'язків між модулями до числа модулів;
- Відношення числа ненормальних виходів з безлічі операторів до загального числа операторів;

L1oop L IF L mod f = N4SV / L mod f * = N * SV / L

Метрики Мак- Кейба - цикломатичне число;

- Цикломатическая складність ;

l (G) = m - n + p n (G) = l (G) +1 = m - n + 2

метрика Чепена

- Міра труднощі розуміння програм на основі вхідних і вихідних даних;

H = 0.5T + P +2 M +3 C

метрика Шнадевіда - Число шляхів в керуючому графі

S = S Pi Ci

метрика Майерса

- Інтервальна міра ;

[n 1 ¸ n 2]

метрика Хансена - Пара (цикломатичне число , число операторів)

{ n , N }

метрика Чена

- Топологічна міра Чена ;

M (G) = (n (G) , N , Q0)

метрика Вудворда

- Вузлова міра (число вузлів передач управління);

Y x

метрика Кулика

- Нормальне число (число найпростіших циклів в нормальній схемі програми);

Norm (P)

метрика Хура

- Цикломатичне число мережі Петрі , що відбиває керуючу структуру програми ;

l (G * p)

Метрики Вітворфа , Зулевского

- міра складності потоку керування
- міра складності потоку даних;

g (P) W (P)

метрика Петерсона

- Число багатовхідних циклів;

Nm 1 0 0 p

Метрики Харрісона , Мейджела

- Функціональне число (сума наведених складнощів всіх вершин керуючого графа) ;
- Функціональне ставлення (відношення числа вершин графа до функціонального числу) ;
- Регулярні вирази (число операндів , операторів і дужок у регулярному виразі керуючого графа програми);

f1 = S c 1 f * = N c 1 / f1 p (G) = N + L + Sk

метрика Пивоварського

- Модифікована цикломатическая міра складності;

N (G) = n * (G) + S Pi

метрика Пратта

- Тестирующая міра ;

Test (Pr)

метрика Кантоні

- Характеристичні числа поліномів , що описують керуючий граф програми ;

PCN *

Метрика Мак- Клур

- Міра складності , заснована на числі можливих шляхів виконання програми , числі керуючих конструкцій і змінних ;

C (V) = D (V) ‘ J (V) / N

метрика Кафур

- Міра на основі концепції інформаційних потоків;

I (G)

Метрика Схуттса , Моханти

- Ентропійні заходи ;

e (G)

метрика Коллофело

- Міра логічної стабільності програм;

h (G)

Метрика Зольновского , Сіммонса , Тейер Зважена сума різних індикаторів : - (Структура , взаємодія , обсяг , дані) ; - (Складність інтерфейсу , обчислювальна складність , складність введення / виводу, читабельність) ;

å (a , b , g , n)

å (c , C , u , p)

метрика Берлінгер

- Інформаційна міра ;

I (R) = m (F * (R) ‘ F- (R)) 2

метрика Шумана

- Складність з позиції статистичної теорії мови ;

X (Y)

метрика Янгера

- Логічна складність з урахуванням історії обчислень;

складність проектування насиченість коментарями Число зовнішніх звернень число операторів

L (w) Cc = å log2 (i + 1) [å n Cxy (n)] X = K / C Ci L1

ПРОГНОЗ МОДЕЛІ

моделі Холстеда

- Прогноз системних ресурсів;
- Прогноз числа помилок.

Модель фірми IBM Модель цілому моделі зв’язності Сплайн -модель

P = 3 / 8 (Ra - 1) ‘ 2Ra B = Nlog2 n / 3000 B = 23M1 + M1 0 B = 21.1 + 0.1 V + COMP (S) Pij = 0.15 (Si + Sj) + 0.7 Cij Pij = $\frac{1}{2}$ å li (D Zij2 + D Nij2) ln (D Zij2 + D Nij2) + a + b Zi + g N1

ОЦІНОЧНІ МОДЕЛІ

Джелінскі - Моранді

R (t) = e - (T - 1 + 1) Ft

Вейса - Байеса

R1 (t) = ò ò e - l - (i -1) F) t Y (l , F/t1 , ... , ti - 1) dl dΦ

Шика - Волвертона

R1 (t) = e - F (N - 1 + 1) ti2 / 2

Літтвуда

R1 (t) = (b + t / b + t + t) - F (N - i + 1) a

Нельсона

Rj (t) = exp { åln (1 - Pj) }

Халецького

Rj (t) = Pμ - a (1 - g nj) / nj

модель налагодженості

Rj (t) = Pμ - r fj (t , l , p)

мозаїчна модель

Rj (t) = 1 - b (a - wj - 1)

У таблиці представлені різноманітні метрики складності ПЗ для різних форм їх подання , моделі прогнозуючі хід розвитку процесів розробки ПЗ та імовірнісні моделі з оцінки надійності. Коротко розглянемо метрики складності. Однією з основних цілей науково -технічної підтримки є зменшення складності ПЗ. Саме це дозволяє знизити трудомісткість проектування , розробки , випробувань і супроводу , забезпечити простоту і надійність виробленого ПЗ. Цілеспрямоване зниження складності ПЗ являє собою багатокрокового процедуру і вимагає попереднього дослідження існуючих показників складності , проведення їх класифікації та співвіднесення з типами програм та їх місцем розташування в життєвому циклі.

Теорія складності програм орієнтована на управління якістю ПЗ і контроль її еталонної складності в період експлуатації. В даний час різноманіття показників (у тій чи іншій мірі описують складність програм) настільки велике , що для їх вживання потрібно попереднє упорядкування . У ряді випадків задовольняються трьома категоріями метрик складності. Перша категорія визначається як словникова метрика , заснована на метричних співвідношеннях Холстеда , цикломатическая заходи Мак- Кейба і вимірах Тейер . Друга категорія орієнтована на метрики зв'язків, що відображають складність відносин між компонентами системи - це метрики Уіна і Вінчестера. Третя категорія включає семантичні метрики , пов'язані з архітектурним побудовою програм та їх оформленням .

Відповідно до іншої класифікації , показники складності діляться на дві групи: складність проектування і складність функціонування . Складність проектування, яка визначається розмірами програми , кількістю оброблюваних змінних , трудомісткістю і тривалістю розробки та ін факторами , аналізується на основі трьох базових компонентів : складність структури програми , складність перетворень (алгоритмів) , складність даних. У другу групу показників віднесені тимчасова , програмна й інформаційна складності , що характеризують експлуатаційні якості ПЗ.

Існує ще ряд підходів до класифікації заходів складності , проте вони , фіксуючи приватні сторони досліджуваних програм , не дозволяють (нехай з великим допущенням) відобразити загальне , то , чиї виміри можуть лягти в основу виробничих рішень .

Загальним , инвариантно властивим будь-якому ПО (і пов'язаної з його коректністю) , є його СТРУКТУРА . Важливо пов'язати це обставина з певним значенням структурної складності в сукупності заходів складності ПЗ. І більше того , при аналізі структурної складності доцільно обмежитися тільки її топологічними заходами , тобто заходами , в основі яких лежать топологічні характеристики граф - моделі програми . Ці заходи задовольняють переважній більшості вимог, що пред'являються до показників : спільність застосовності , адекватність розглянутого властивості , істотність оцінки , спроможність, кількісне вираження , відтворюваність вимірювань , мала трудомісткість обчислень , можливість автоматизації оцінювання .

Саме топологічні міри складності найчастіше застосовуються у фазі досліджень , формує рішення з управління виробництвом (в процесах проектування , розробки і випробувань) і становлять доступний і чутливий еталон готової продукції , контроль якого необхідно регулярно здійснювати в період її експлуатації.

Першою топологічної мірою складності є цикломатическая міра Мак- Кейба . В її основі лежить ідея оцінки складності ПЗ за кількістю базисних шляхів в її керуючому графі , тобто таких шляхів , компонуючи які можна отримати всілякі шляхи з входу графа в виходи . Цикломатичне число l (G) орграфа G з n - вершинами , m - дугами і p - компонентами зв'язності є величина l (G) = m - n + p .

Має місце теорема про те , що число базисних шляхів в орграфе одно його цикломатическая числу , збільшеному на одиницю . При цьому , цикломатическая складністю ПО P з керуючим графом G називається величина n (G) = l (G) +1 = m - n + 2 . Практично цикломатическая складність ПО дорівнює числу предикатів плюс одиниця , що дозволяє обчислювати її без побудови керуючого графа простим підрахунком предикатів . Дана міра відображає психологічну складність ПЗ.

До достоїнств заходи відносять простоту її обчислення і повторюваність результату , а також наочність і змістовність інтерпретації. Як недоліки можна відзначити: нечутливість до розміру ПО , нечутливість до зміни структури ПО , відсутність кореляції зі структурованістю ПО , відсутність відмінності між конструкціями Розвилка і Цикл , відсутність чутливості до вкладеності циклів . Недоліки цикломатическая заходи призвело до появи її модифікацій , а також принципово інших заходів складності.

Дж. Майерс запропонував як міри складності інтервал [n 1 , n 2] , де n 1 - цикломатическая міра , а n 2 - число окремих умов плюс одиниця . При цьому , оператор DO вважається за одну умову , а CASE cn - исходами за n - 1 - умов. Введена міра отримала назву інтервального заходом.

У. Хансену належить ідея брати в якості міри складності ПО пару { цикломатическая число , число операторів } . Відома топологічна міра Z (G) , чутлива до структурованості ПЗ. При цьому , вона Z (G) = V (G) (дорівнює цикломатическая складності) для структурованих програм і Z (G) > V (G) для неструктурованих . До варіантів цикломатическая міри складності відносять також міру M (G) = (V (G) , C , Q) , де C - кількість умов , необхідних для покриття керуючого графа мінімальним числом маршрутів , а Q - ступінь зв'язності структури графа програми та її протяжність .

До заходів складності , враховує вкладеність керуючих конструкцій , відносять тестуючу міру M і міру Харрісона - Мейджела , що враховують рівень вкладеності і протяжності ПО , міру Пивоварського - цикломатическая складність і глибину вкладеності , і міру Мак- Клур - складність схеми розбиття ПО на модулі з урахуванням вкладеності модулів і їх внутрішньої складності .

Функціональна міра складності Харрісона - Мейджела передбачає приписування кожній вершині графа своєї власної складності (первинної) і розбиття графа на сфери впливу предикатних вершин. Складність сфери називають наведеною і складають її з первинних складнощів вершин , що входять в сферу її впливу , плюс первинну складність самої предикатной вершини . Первинні складності обчислюються всіма можливими способами. Звідси функціональна міра складності ПО є сума наведених складнощів всіх вершин керуючого графа.

Міра Пивоварського ставить метою врахувати в оцінці складності ПО відмінності не тільки між послідовними і вкладеними керуючими конструкціями , а й між структурованими і неструктурованими програмами. Вона виражається відношенням N (G) = n * (G) + S Pi , де n * (G) - модифікована цикломатическая складність , обчислена так само, як і V (G) , але з однією відмінністю : оператор

CASE з n - виходами розглядається як один логічний оператор , а не як n - 1 операторів. Pi - глибина вкладеності i - тієї предикатной вершини .

Для підрахунку глибини укладення предикатних вершин використовується число сфер впливу. Під глибиною вкладеності розуміється число всіх сфер впливу предикатів , які або повністю утримуватися в сфері розглянутої вершини , або перетинаються з нею. Глибина вкладеності збільшується за рахунок вкладеності не самих предикатів , а сфер впливу. Порівняльний аналіз цикломатическая і функціональних заходів з обговорюваної для десятка різних керуючих графів програми показує , що при нечутливості інших заходів цього класу , міра Пивоварського зростає при переході від послідовних програм до вкладених і далі до неструктурованих .

Міра Мак- Клур призначена для управління складністю структурованих програм у процесі проектування. Вона застосовується до ієрархічним схемами розбивки програм на модулі , що дозволяє вибрати схему розбиття з меншою складністю задовго до написання програми . Метрикою виступає залежність складності програми від числа можливих шляхів виконання, числа керуючих конструкцій і числа змінних (від яких залежить вибір шляху) . Методика розрахунку складності по Мак- Клур чітко орієнтована на добре структуровані програми .

Тестуючої мірою M називається міра складності , яка задовольняє таким умовам:

1. Міра складності простого оператора дорівнює 1 ;
2. M ({ F1 ; F2 ; ; Fn }) = Ein M (Fi);
3. M (IF P THEN F1 ELSE F2) = 2 MAX (M (F1) , M (F2)) ;
4. M (WHILE P DO F) = 2 M (F).

Міра зростає з глибиною вкладеності і враховує протяжність програми . До тестирующей міру близько примикає міра на основі регулярних вкладень . Ідея цієї міри складності програм полягає в підрахунку сумарного числа символів (операндів , операторів , дужок) в регулярному виразі з мінімально необхідним числом дужок , що описує керуючий граф програми . Всі заходи цієї групи чутливі до вкладеності керуючих конструкцій і до протяжності програми . Однак зростає рівень трудомісткості обчислень.

Розглянемо заходи складності , що враховують характер розгалужень . В основі вузловий заходи Вудворда , Хедлі лежить ідея підрахунку топологічних характеристик потоку управління . При цьому , під вузловий складністю розуміється число вузлів передач управління . Дана міра відстежує складність лінеаризації програми і чутлива до структуризації (складність зменшується). Вона застосовна для порівняння еквівалентних програм , переважніше заходи Холстеда , але по спільності поступається міру Мак- Кейба .

Топологічна міра Чена висловлює складність програми числа перетинів кордонів між областями , утвореними блок - схемою програми . Цей підхід застосовується лише до структурованим програмами , що допускає лише послідовне з'єднання керуючих конструкцій . Для неструктурованих програм міра Чена істотно залежить від умовних і безумовних переходів . У цьому випадку можна вказати верхню і нижню межі міри. Верхня - є m +1 , де m - число логічних операторів при їх гніздовий вкладеності . Нижня - дорівнює 2 . Коли керуючий граф програми має тільки одну компоненту зв'язності , міра Чена збігається з цикломатическая мірою Мак- Кейба .

Метрики Джілбі оцінюють складність графоорієнтірованних модулів програм відношенням числа переходів за умовою до загального числа виконуваних операторів . Добре зарекомендувала себе метрика , що відноситься число міжмодульних зв'язків до загального числа модулів. Названі метрики використовувалися для оцінки складності еквівалентних схем програм , особливо схем Янова .

Використовуються також міри складності , що враховують історію обчислень , характер взаємодії модулів і комплексні заходи .

Сукупність цикломатическая заходів придатна для оцінювання складності первинних формалізованих специфікацій , які задають в сукупності вихідні дані , цілі та умови побудови шуканого ПЗ. Оцінка

цієї первинної програми або порівняння декількох альтернативних її варіантів дозволить спочатку гармонізувати процес розробки ПЗ та від стартової точки контролювати і управляти його поточної результуючої складністю.

Г.Майерс : Надійність програмного забезпечення

Оскільки обробка даних зачіпає наше життя все більшою мірою , помилки ЕОМ можуть тепер мати такі наслідки , як нанесення матеріального збитку , порушення секретності і багато інших , включаючи смерть.

Що таке помилка в програмному забезпеченні і що таке надійність програмного забезпечення? Важливо домовитися про стандартний визначенні , щоб уникнути таких ситуацій , коли користувач стверджує, що виявив в системі помилку , а розробник відповідає: "Ні , система так і була задумана " .

Що таке помилка ?

Приклад : Система раннього виявлення балістичних снарядів Ballistic Missile Early Warning System повинна спостерігати за об'єктами, що рухаються у напрямку до США , і , якщо об'єкт не пізнаний , почати послідовність захисних заходів - від спроб встановити з об'єктом зв'язок до перехоплення і знищення. Одна з ранніх версій системи помилково брала піднімається над горизонтом Місяць за снаряд , що летить над Північною півкулею . Чи є це помилкою? З точки зору користувача (Міністерства оборони США) - так. З точки зору розробника системи - можливо , й ні. Розробник може наполягати на тому , що у відповідності зі специфікаціями захисні дії повинні бути розпочаті по відношенню до будь-якого рухається об'єкту , що з'явився над горизонтом і не пізнаний як мирний літальний апарат.

Справа в тому , що різні люди по- різному розуміють , що таке помилка в програмному забезпеченні.

Програмне забезпечення містить помилку , якщо :

1. його поведінка не відповідає специфікаціям. Недоліки: неявно передбачається , що специфікації коректні. Це якщо й буває справедливим , то рідко ; підготовка специфікацій - один з основних джерел помилок. Якщо поведінка програмного продукту не відповідає його специфікаціям , помилка , ймовірно , є. Однак , якщо система веде себе у відповідності зі специфікаціями , ми не можемо стверджувати , що вона не містить помилок.

2. його поведінка не відповідає специфікаціям при використанні у встановлених при розробці межах. Це визначення ще гірше першого . Якщо система випадково використовується в непередбачуваної ситуації , її поведінка повинна залишатися розумним. Якщо це не так , вона містить помилку. Наприклад , авіаційна диспетчерська система , згідно специфікаціям , повинна управляти рухом до 200 літаків одночасно. Але одного разу , в районі з'явився 201 літак. Якщо поведінка системи нерозумно - скажімо , вона забуває про один з літаків або виходить з ладу , система містить помилку , хоча і використовується поза межами , встановлених при проектуванні .

3. програмне забезпечення поводиться не відповідно до офіційної документацією та поставленими користувачеві публікаціями. А якщо помилки містяться і в програмі і в публікаціях ? Або якщо в керівництві описана тільки очікувана і планована робота з системою. Наприклад , написано: "Щоб отримати те-то , натисніть один раз те-то " . Припустимо , що користувач випадково два рази натискає те-то і система виходить з ладу , тому що її розробники не передбачили такої ситуації. Система , очевидно , містить помилку , але веде себе відповідно з публікаціями .

4. система не здатна діяти відповідно з вихідним контрактом та переліком вимог користувача. Це твердження теж не позбавлене недоліків , оскільки письмові вимоги користувача рідко деталізовані настільки, щоб описувати бажану поведінку програмного забезпечення при всіх мислимих обставин.

Остаточне визначення : У програмному забезпеченні є помилка , якщо воно не виконує того , що користувачеві розумно від нього очікувати. Відмова програмного забезпечення - це прояв помилки в ньому . Що таке надійність?

Згідно відомим визначенням , надійність є ймовірність того , що при функціонуванні системи протягом деякого періоду часу не буде виявлено жодної помилки.

Основний недолік такого визначення - це те , що в ньому не враховано відмінність між помилками різних типів. Наприклад , розглянемо авіаційну диспетчерську систему з двома помилками в програмному забезпеченні : через однієї втрачається слід літака , а інша полягає в тому , що в повідомленні оператору неправильно друкується одне слово (наприклад , ТРАНСАТТЛАНТІЧЕСКІЙ замість ТРАНСАТЛАНТИЧНИЙ) . За своїми наслідками ці помилки далеко не однакові , тому надійність повинна бути визначена як функція не тільки частоти помилок , але і їх серйозності. отже : Н адійність програмного забезпечення є ймовірність його роботи без відмов протягом певного періоду часу , розрахована з урахуванням вартості для користувача кожного відмови.

Таким чином , надійність програмного забезпечення є функцією впливу помилок на користувача системи ; вона не обов'язково прямо пов'язана з оцінкою "зсередини" програмного забезпечення. Навіть великий прорахунок в проектуванні може виявитися не дуже помітним для користувача. З іншого боку , начебто б тривіальна помилка може мати катастрофічні наслідки. Наприклад , перший запуск космічного корабля на Венеру зазнав невдачі через те , що в операторі DO програми на Фортране була пропущена кома .

Надійність і вартість програмного забезпечення.

Типовий розподіл вартості програмного забезпечення:

17 % - Проектування ;

8 % - Програмування ;

25 % - Тестування ;

50 % - Супровід .

Висока вартість програмного забезпечення - багато в чому наслідок низької надійності. При збільшенні продуктивності програміста (якщо вимірювати її тільки швидкістю розробки та кодування програми) вартість істотно не зменшується. Спроби збільшити продуктивність програміста можуть у деяких випадках навіть підвищити вартість . Найкращий шлях скорочення вартості - у зменшенні вартості його тестування і супроводу. А це може (Під супроводом розуміється будь-яке продовження роботи з програмним продуктом , такі як зміни , доповнення тощо з метою забезпечення його подальшої працездатності та відповідності його вимогам часу) . А це може бути досягнуто не за рахунок інструментів , покликаних збільшити швидкість програмування , а лише в результаті розробки засобів, що підвищують коректність і чіткість при створенні програмного забезпечення. Основні принципи проектування .

Всі принципи та методи забезпечення надійності відповідно до їх метою можна розбити на чотири групи:

1. Попередження помилок
2. Виявлення помилок
3. Виправлення помилок
4. Забезпечення стійкості до помилок

Системи дистанційного банківського обслуго-вування (ДБО) є критичним елементом банківської інфраструктури, вони у значній мірі визначають конкурентоспроможність банку в сучасних умовах глобалізації ринку банківських послуг. Банки вимушені розробляти і вдосконалювати свої системи ДБО надзвичайно швидкими темпами, залучаючи як внутрішні IT-відділи, так і сторонніх розробників, як правило, у формі аутсорсингу. Крім того, особли-во актуальним постає питання залучення відкритого (opensource) програмного забезпечення (ПЗ), яке є надзвичайно привабливим з комерційної точки зору за рахунок низьких початкових витрат, однак містить деякі потенційні ризики, серед яких принциповим, на наш погляд, є підвищений ризик вразливості банківських рішень до різного роду зловмисних

дій за рахунок доступності вихідного коду для дослідження з метою планування атак. Таким чином, системи ДБО вимагають розробки певних науково обґрунтованих рішень, що могли б допомогти проконтролювати їх внутрішню будову і дозволити зменшити потенційні ризики у напрямку надійності і безпеки систем.

4.2 Аналіз алгоритмів виявлення залежностей між потенційно небезпечними дефектами реакції програм

В еру стрімкого розвитку IT, ефективність кібернетичного впливу залежить не тільки від можливості знайти вразливість, але й від часових характеристик, адже вразливість, яка сьогодні знайдена, завтра може бути вже усунутою. Для забезпечення кращих часових показників необхідно провести правильний розподіл робочих ресурсів для аналізу та розробки засобів впливу - і в ручну таке завдання зайняло б недопустиму кількість часу. Тому використання метрк коду в цьому випадку може допомогти у виборі тих участків, областей коду, де ймовірність успішного впровадження більша.

Тому з однієї сторони код потрібно аналізувати на предмет наявності потенційно-небезпечних дефектів коду, а з іншої - вираховувати його метрики та спрямовувати зусилля в тому напрямку, в якому для цього сприяють і самі властивості, характеристики вихідного тексту.

Як відомо дефект переповнення буферу не завжди може бути використаним для використання в цілях кібернетичного впливу. Але можна припустити, що в слабо структурованому коді, який містить запутану логіку, велику кількість переходів, розгалужень та змінних набагато легше допустити критичну помилку, яка надасть можливість провести кібернетичний вплив від зацікавленої сторони.

Тому пропоную аналізувати код за допомогою наступних метрик:

- метрики Холстеда (розмірність коду, словника)
- метрики Маккейба (цикломатична складність)
- метрики Джилба (складність розгалуження коду)

4.2.1 Детальний опис метрик:

4.2.2 Метрика Холстеда

До групи оцінок розміру програм можна віднести також метрику Холстеда . За базу прийнятий підрахунок кількості операторів і операндів використовуються у програмі , тобто визначення розміру програми .

Основу метрики Холстеда складають чотири вимірювані характеристики програми : h1 - число унікальних операторів програми , включаючи символи - роздільники , імена процедур і знаки операцій (словник операторів) ; h2 - число унікальних операндів програми (словник операндів) ; N1 - загальне число операторів в програмі N2 - загальне число операндів в програмі.

Спираючись на ці характеристики , одержувані безпосередньо при аналізі вихідних текстів програм , Холстед вводить такі оцінки

Словник програми h = h1 + h2

Довжину програми N = N1 + N2

Обсяг програми V = N log2 h

Сенс оцінок h і N досить очевидний , тому докладно розглянемо тільки характеристику V.

Кількість символів , що використовуються при реалізації деякого алгоритму , визначається в числі інших параметрів і словників програми h , що представляє собою мінімально необхідне число символів , що забезпечують реалізацію алгоритму .

Далі Холстед вводить h * - теоретичний словник програми, тобто словниковий запас, необхідний для написання програми з урахуванням того , що необхідна функція вже реалізована в даній мові і, отже , програма зводиться до виклику цієї функції. Наприклад , згідно Холстеду можливе здійснення процедури виділення простого числа могло б виглядати так :

CALL SIMPLE (X , Y) ,

де Y- масив чисельних значень , що містять шукане число X.

Теоретичний словник в цьому випадку буде складатися з

n1 * : { CALL , SIMPLE (...) } n1 * = 2 ;

n2 * : { X , Y } , h2 * = 2 ;

а його довжина , обумовлена як

h * = h1 * + h2 * дорівнюватиме 4 .

Використовуючи h * , Холстед вводить оцінку V * : V * = h * log2h * ,

за допомогою якої описується потенційний обсяг програми , соотвествующий максимально компактно реалізує даний алгоритм.

4.2.3 Метрика Маккейба

Друга найбільш представницька група оцінок складності програм - метрики складності потоку керування програм. Як правило , за допомогою цих оцінок оперують або щільністю керуючих переходів усередині програм, або взаємозв'язками цих переходів .

І в тому і в іншому випадку стало традиційним уявлення програм у вигляді керуючого орієнтованого графа G (V , E) , де V - вершини , відповідні операторам , а E - дуги , відповідні переходам . У дузі (U , V) - вершина V є вихідною , а U - кінцевої . При цьому U безпосередньо випливає V , а V безпосередньо передує U. Якщо шлях від V до U складається більш ніж з однієї дуги , тоді U слід за V , а V передує U.

Вперше графічне представлення програм було запропоновано МакКейб . Основний метрикою складності він пропонує вважати цикломатическая складність графа програми , або , як ще називають , цикломатичне число МакКейб , що характеризує трудомісткість тестування програми .

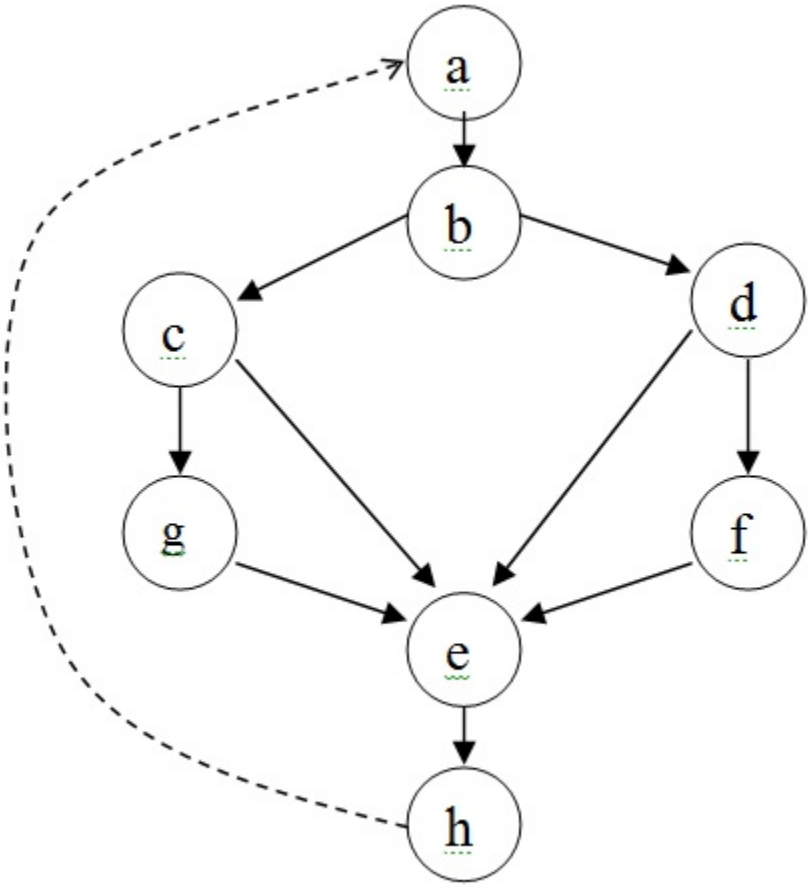
Для обчислення цикломатическая числа МакКейб Z (G) застосовується формула

Z (G) = l - v +2 p ,

де l - число дуг орієнтованого графа G ; v - число вершин ; p -число компонентів зв'язності графа.

Число компонентів зв'язності графа можна розглядати як кількість дуг , які необхідно додати для перетворення графа сільносвязний . Сільносвязний називається граф , будь-які дві вершини якого взаємно досяжні. Для графів коректних програм, тобто графів , що не мають недосяжних від точок входу дільниць і « висячих » входу і виходу , сільносвязний граф , як правило , виходить шляхом замикання однієї вершини , що позначає кінець програми на вершину , що позначає точку входу в цю програму.

По суті Z (G) визначає число лінійно незалежних контурів у сільносвязний графі . Інакше кажучи , цикломатичне число МакКейб показує необхідне число проходів для покриття всіх контурів Сільносвязанная графа або кількість тестових прогонів програми , необхідних для вичерпного тестування за критерієм « працює кожна гілка» .



Для програми цикломатичне число при l = 10 , v = 8 , n = 1 визначиться як Z (G) = 10-8 +2 = 4 .

Таким чином, є Сільносвязанная граф з чотирма лінійно незалежними контурами :

a - b - c - g - e - h - a ;

a - b - c - e - h - a ;

a - b - d - f - e - h - a ;

a - b - d - e - h - a ;

Розглянемо метрику складності програми , що отримала назву « підрахунок точок перетину « , авторами якої є М Вудвард , М Хенель і Д Хидли . Метрика орієнтована на аналіз програм , при створенні яких використовувалося неструктурні кодування на таких мовах , як мова асемблера і фортран . Вводячи цю метрику , її автори прагнули оцінити взаємозв'язку між фізичними місцеположеннями керуючих переходів .

Структурний кодування припускає використання обмеженого безлічі керуючих структур в якості первинних елементів будь-якої програми . У класичному структурному кодуванні , що базується на роботах професора Ейндховенського технологічного університету (Нідерланди) Е. Дейкстри , оперують тільки трьома такими структурами : проходженням операторів , розвилкою з операторів , циклом над оператором. всі ці різновиди зображуються найпростішими планарними графами програм. За правилами структурного кодування будь-яка програма складається шляхом вибудовування ланцюжків з 3х згаданих структур або приміщення однієї структури в іншу. Ці операції не порушують планарности графа всієї програми .

У графі програми , де кожному оператору відповідає вершина , тобто не виключені лінійні ділянки

, при передачі управління від вершини a до b номер оператора a дорівнює min (a , b) , а номер оператора b - max (a , b) . Тоді перетин дуг з'являться , якщо

min (a , b) < min (p , q) < max (a , b) & max (p , q) > max (a , b) |

Min (a , b) < max (p , q) < max (a , b) & min (p , q) < min (a , b) .

Іншими словами , точка перетину дуг виникає у разі виходу управління за межі пари вершин (a , b) .

Кількість точок перетину дуг графа програми дає характеристику неструктурованості програми.

4.2.4 Метрика Джилба

Однією з найбільш простих , але досить ефективних оцінок складності програм є метрика Т. Джілбі , в якій логічна складність програми визначається як насиченість програми виразами IF_THEN_ELSE . При цьому вводяться дві характеристики :

CL - абсолютна складність програми , що характеризується кількістю операторів умови; cl - відносна складність програми , що характеризується насиченістю програми операторами умови , тобто cl визначається як відношення CL до загального числа операторів. Використовуючи метрику Джілбі , її доповнили ще однією складовою , а саме характеристикою максимального рівня вкладеності оператора CLI , що дозволило застосувати метрику Джілбі до аналізу циклічних конструкцій.

Великий інтерес представляє оцінка складності програм за методом граничних значень .

Введемо кілька додаткових понять , пов'язаних з графом програми .

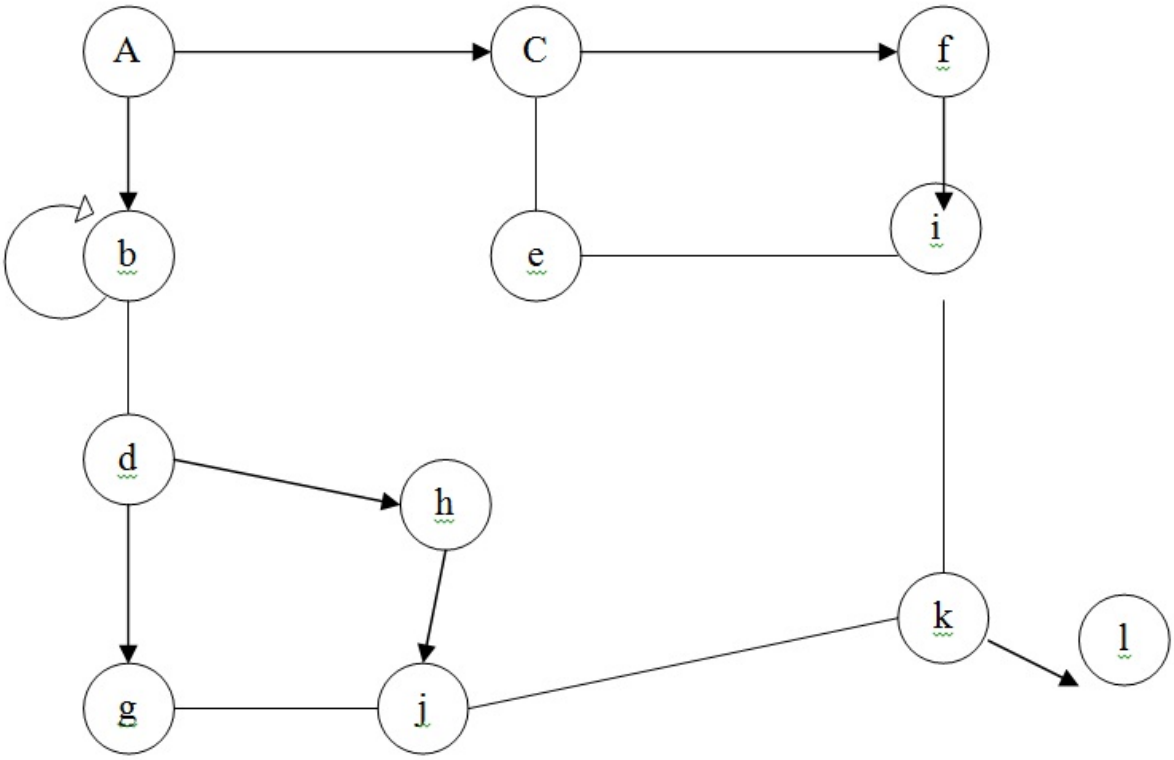
Нехай G = (V , E) - орієнтований граф програми з єдиною початковою і єдиною окнечной вершинами. У цьому графі число входять до вершину дуг називається негативною ступенем вершини , а число що виходять з вершини дуг - позитивної ступенем вершини . Тоді набір вершин графа можна розбити на дві групи:

вершини у яких позитивна ступінь <= 1 ; вершини у яких позитивна ступінь > = 2 . Вершини першої групи назвемо приймаючими вершинами , а вершини другої групи - вершинами відбору.

Для отримання оцінки за методом граничних значень необхідно розбити

граф G на максимальне число подграфов G ' , що задовольняють таким умовам:

вхід в подграф здійснюється тільки через вершину відбору; кожен подграф включає вершину (звану нижньою межею подграфа) , в яку можна потрапити з будь-якої іншої вершини подграфа . Наприклад , вершина відбору поєднана сама з собою дугою петлею , утворює подграф .



Число вершин , що утворюють такий подграф , одно скоригованої складності вершини відбору.

Кожна приймаюча вершина має скориговану складність , рівну 1 , крім кінцевої вершини , скоригована складність якої дорівнює 0 . Скориговані складності всіх вершин графа G підсумовуються , утворюючи абсолютну граничну складність програми . Після цього визначається відносна гранична складність програми :

S0 = 1 - (v - 1) / Sa ,

де S0 - відносна гранична складність програми ; Sa - абсолютна гранична складність програми , v - загальне число вершин графа програми .

Таким чином , відносна складність програми дорівнює

S0 = 1 - (11 / 25) = 0,56 .

Інша група метрик складності програм - метрика складності потоку даних , тобто використання , конфігурації і розміщення даних в програмах.

Пара « модуль - глобальна змінна» позначається як (p , r), де p - модуль , що має доступ до глобальної змінної r . Залежно від наявності в програмі реального обігу до змінної r формуються два типи пар « модуль - глобальна змінна» : фактичні і можливі . Можливе звернення до r за допомогою p показує , що область існування r включає в себе p .

Характеристика Aup говорить про те , скільки разів модулі Up дійсно отримали доступ до глобальних змінних , а число Pup - скільки разів вони могли б отримати доступ.

Відношення числа фактичних звернень до можливих визначається

Rup = Aup / Pup

Ця формула показує наближену ймовірність посилання довільного модуля на довільну глобальну змінну. Очевидно , чим вище ця вірогідність , тим вище ймовірність « несанкціонованого » зміни якої-небудь змінної , що може істотно ускладнити роботи, пов'язані з модифікацією програми .

Покажемо розрахунок метрики « модуль - глобальна змінна» . Нехай у програмі є три глобальні змінні і три підпрограми. Якщо припустити , що кожна підпрограма має доступ до кожної з змінних , то ми отримаємо дев'ять можливих пар , тобто Pup = 9 . Далі нехай першим підпрограма звертається до однієї змінної , другий - двом , а третя не звертається ні до однієї змінної. Тоді Aup = 3 , Rup = 3 /9.

Ще одна метрика складності потоку даних - Спен .

Визначення Спен грунтується на локалізації звернення до даних всередині кожної програмної секції.

Спен - це число тверджень , які містять даний ідентифікатор , між його першим і останнім появою в тексті програми . Ідентифікатор, що з'явився n раз , має Спен , рівний n - 1 .

Спен визначає кількість контролюючих тверджень , що вводяться в тіло програми при побудові траси програми з цього ідентифікатором в процесі тестування і налагодження.

4.3 Алгоритм пошуку залежностей ПНДРП на основі екстраполяції метричних характеристик вихідних текстів програм для побудови дерева атак

Розглянемо програмний продукт в якості множини вразливостей:

$Vuln = Vuln_1, Vuln_2, ..., Vuln_N$

Маючи вищенаведені характеристики, та множину потенцыйних вразливостей пропоную створити наступну модель вибору потенційно небезпечних дефектів реакції програм для кібернетичного впливу:

$P_{Vuln_i} = \frac{V_i Z(G)_i}{Z(G)\frac{V}{C_v}} Rup_i,$

- V - Обсяг програми
- Vi - обсяг підпрограми
- Cv - кікькість потенційних вразливостей
- Z(G)i - цикломатична складність підпрограми, в якій знаходиться дефект
- Z(G) - цикломатична складність всього коду досліджуваного проекту
- Rup - кількість звернень потенційно-вразливої ділянки коду до глобальних змінних

Розглянемо кожну властивість поданих метрик в контексті відображення наявності можливості викори

- Обсяг програми - обсяг програми напряму впливає на кількість помилок в ній
- Аналогічним чиним від обсягу підпрограми, в якій знаходиться потенційна вразливість залежить можливість її використання
- Від цикломатичної складності залежить наскільки просто буде проаналізувати логіку коду і швидко розібратись, як саме можна використати дефект
- А від кількості звернень до потенційно-вразливої ділянки коду можна зробити висновок, як локальні дані між собою звязані і чи можливо здійснити вплив на певні управляючі дані через суміжні

Дана ймовірність буде наближеною та неточною, але якщо набір таких ймовірностей збільшувати і розглянути їх як протабульовану функцію $f(V, V_i, C_v, Z(G)_i, Z(G), Rup) = P_{vuln_i}$ то можна спробувати робити прогноз наявності вразливих ділянок при аналізі нового вихідного тексту на основі його метрик коду - що дозволить побудувати ефективне дерево атак.

4.3.1 Дерева атак

Дерева атак - це діаграми, що демонструють, як може бути атакована мету. Дерева атак використовуються в безлічі областей. В області інформаційних технологій вони застосовуються, щоб описати потенційні загрози комп'ютерній системі і можливі способи атаки , реалізують ці загрози. Однак, їх використання не обмежується аналізом звичайних інформаційних систем. Вони також широко використовуються в авіації і обороні для аналізу ймовірних загроз, пов'язаних зі стійкими до спотворень електронними системами. (Наприклад на авіонику військових літальних засобів).

Збільшується застосування дерев атак в комп'ютерних системах контролю (особливо пов'язаних з енергетичними мережами). [2] Також дерева атаки використовуються для розуміння загроз, пов'язаних з фізичними системами.

Деякі з найбільш ранніх описів дерев атак знайдені в доповідях і статтях Брюса Шнайера [3] , технічного директора Counterpane Internet Security. Шнайер був безпосередньо залучений в розробку концептуальної моделі дерев атаки і зіграв важливу роль в її поширенні. Тим не менш, в деяких ранніх опублікованих статтях по деревах атак [4] висловлюються припущення про залученість Агентства Національної Безпеки в початковий етап розробки.

Дерева атак дуже схожі на дерева загроз. Дерева загроз були розглянуті в 1994 роки Едвардом Аморосо. [5]

Дерева атак це мультирівневих діаграми, що складаються з одного кореня, листя і нащадків. Будемо розглядати вузли знизу вгору. Дочірні вузли це умови, які повинні виконуватися, щоб батьківський вузол також перейшов в справжній стан. Коли корінь переходить у справжній стан, атака успішно завершена. Кожен вузол може бути приведений у справжній стан тільки його прямими нащадками.

Вузол може бути дочірнім для іншого вузла, в цьому випадку, логічно, що для успіху атаки потрібно кілька кроків. Наприклад, уявіть клас з комп'ютерами, де комп'ютери прикріплені до парт. Щоб вкрасти один з них необхідно або перерізати кріплення, або відкрити замок. Замок можна відкрити відмичкою або ключем. Ключ можна отримати шляхом погроз його власнику, через підкуп власника або ж просто вкрасти його. Таким чином, можна намалювати чотирирівневої дерево атаки, де одним із шляхів буде: Підкуп власника ключа-Отримання ключа-Відмикання замку-Винос комп'ютера.

Також слід враховувати, що атака, описана у вузлі може зажадати, щоб одна або декілька з безлічі атак, описаних в дочірніх вузлах були успішно проведені. Вище ми показали дерево атаки тільки зі зв'язком типу АБО між нащадками вузла, але умова І також може бути введено, наприклад в класі є електронна сигналізація, яка повинна бути відключена, але тільки в тому випадку, якщо ми вирішимо перерізати кріплення. Замість того, щоб робити відключення сигналізації дочірнім вузлом для перерізання кріплення, обидва завдання можна просто логічно підсумувати, створивши шлях (Відключення сигналізації І Перерізання кріплення)-Винос Комп'ютера.

Дерева атак також пов'язані із створенням дерева помилок. [6] Метод побудови дерева помилок використовує булеві вирази для створення умов, при яких дочірні вузли забезпечують виконання батьківських вузлів.

Включає апріорні ймовірності в кожен вузол, можливо зробити підрахунок ймовірностей для вузлів, що знаходяться вище по правилу Байеса . Однак, в реальності, точні оцінки ймовірності або недоступні, або занадто дорогі для обчислення. У разі динамічної комп'ютерної безпеки (тобто з урахуванням атакуючих) випадкові події не є незалежними, отже, прямий Байєсівський аналіз не підходить.

Так як Байєсовські аналітичні техніки, що використовуються в аналізі дерев помилок не можуть бути правильно застосовані до дерев атак, аналітики використовують інші техніки [7] [8] для визначення

яким шляхом піде даний атакуючий. Ці техніки включають порівняння можливостей атакуючого (час, гроші, навички, обладнання) з ресурсами, що вимагаються для даної атаки. Атаки, які вимагають повної віддачі від атакуючого або навіть знаходяться за межами його можливостей куди менш імовірні, ніж дешеві і прості атаки. Ступінь, в якій атака задовольняє цілям атакуючого також впливає на його вибір. З двох можливих атак зазвичай вибирається та, що більшою мірою задовольняє цілям атакуючого.

Дерева атак можуть стати вкрай складними, особливо при розгляді конкретних атак. Повне дерево атаки може містити сотні або тисячі різних шляхів, всі з яких призводять до успіху атаки. Але навіть при такому розкладі, ці дерева вкрай корисні для визначення існуючих загроз і методів їх запобігання.

Дерева атак можуть бути використані для визначення стратегії забезпечення інформаційної безпеки. Також, слід враховувати, що реалізація цієї стратегії сама по собі вносить зміни в дерево атаки. Наприклад, захистом від комп'ютерних вірусів може служити заборона системного адміністратора безпосередньо змінювати існуючі файли і папки, замість цього вимагаючи використання файлового менеджера. Це додає в дерево атаки використання недоліків чи експлойтів файлового менеджера.

ПРОГРАМНИЙ МОДУЛЬ ВИЯВЛЕННЯ ЗАЛЕЖНОСТЕЙ МІЖ ПОТЕНЦІЙНО-НЕБЕЗПЕЧНИМИ ДЕФЕКТАМИ ПОЧАТКОВОГО ТЕКСТУ ЦІЛЬОВИХ ПРОГРАМ КІБЕРНЕТИЧНОГО ВПЛИВУ

5.1 Структурна схема алгоритму

5.2 Опис інтерфейсу користувача

5.3 Опис програмної реалізації

Розділ 6

ОЦІНКА ЕФЕКТИВНОСТІ ВИРІШЕННЯ ЗАДАЧІ ВИЯВЛЕННЯ ЗАЛЕЖНОСТЕЙ МІЖ ПОТЕНЦІЙНО-НЕБЕЗПЕЧНИМИ ДЕФЕКТАМИ ПОЧАТКОВОГО ТЕКСТУ ЦІЛЬОВИХ ПРОГРАМ КІБЕРНЕТИЧНОГО ВПЛИВУ

6.1 Опис випробування та порівняльний аналіз результатів

6.2 Напрямки удосконалення прототипу системи

ЗАКЛЮЧЕННЯ

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. http://dorlov.blogspot.com/2009/11/blog-post.html

Розділ 9

ДОДАТКИ